Remarks:

Reconsideration of the above referenced application in view of the enclosed amendment and remarks is requested. Claims 2, 23, and 42 have been amended. Existing Claims 1 to 50 remain in the application.

ARGUMENT

Claims 1-4, 9-10, 22-24, 29-30, 42 and 45 are rejected under 35 U.S.C. § 102(b) as being anticipated by USPN 5,245,615 to Treu (hereinafter, "Treu"). This rejection is respectfully traversed and Claims 1-4, 9-10, 22-24, 29-30, 42 and 45 and their progeny are believed allowable based on the following discussion.

MPEP § 706.02 IV states that for anticipation under 35 U.S.C. § 102, the reference must teach every aspect of the claimed invention either explicitly or impliedly. Any feature not directly taught must be inherently present. As a preliminary matter, Applicants respectfully submit that the rejection of Claims 1-4, 9-10, 22-24, 29-30, 42 and 45 is facially deficient because the Examiner has not established a prima facie case of anticipation. As is well-established, in order to establish a prima facie case of unpatentability under 35 U.S.C. § 102, the cited prior art must disclose every limitation of the claims being rejected. Therefore, if even one claim element or limitation is not disclosed by the references, a prima facie case is not established. Additionally, as the Federal Circuit has noted,

> "As adapted to *ex parte* procedure, Graham [v. John Deere Co.] is interpreted as continuing to place the 'burden of proof on the Patent Office which requires it to produce the factual basis for its rejection of an application under sections 102 and 103." (emphasis added)

In re Piasecki, 745 F.2d 1468, 1472, 223 USPQ 785, 788 (Fed. Cir. 1984) (citing *In re Warner*, 379 F.2d 1011, 1016, 154 USPQ 173, 177 (CCPA 1967)). The Examiner thus has the burden of producing a factual basis for his rejection and for establishing anticipation by identifying how each recited claim element is allegedly disclosed by the cited reference. Applicants respectfully submit that the Examiner has failed to establish such a *prima facie* case

and has merely provided bare allegations that the reference anticipates the claims. For example, with respect to the first element of Claims 1, 22, and the last element of amended Claim 42 the Examiner recites the claim element and cites "[Fig. 4 and 5; col. 7, lines 38-52; error logging]." First, the Examiner fails to cite a specific element of the figures. Second, the Examiner cites the description of Figure 5, as below, that does not relate to the recited element:

> "FIG. 5 illustrates the detailed steps for using BIOS to write or store information in error log 88. Before BIOS is called, step 250 stores the error information in buffer 131, such information having been collected and formatted in step 200 (FIG. 4). Step 251 then sets a pointer to such buffer in the ES:DI register of microprocessor 12. Next, the AH and AL registers are respectively set in step 252 for an error log write operation, and a BIOS INT 15H call is made in step 254. BIOS then moves the information from buffer 131 into a BIOS buffer (not shown) and performs a character redundancy check (CRC) on NVRAM 30 to determine if the data therein is valid, in step 258. If the data is invalid, step 176 sets a return code indicating such fact, and step 278 returns to the caller."

The Examiner cites a reference that clearly teaches a method for using a standard BIOS interrupt (15H) to log errors. The Examiner then jumps to the conclusion that this section anticipates the claims. Applicants' recited claims explicitly require that an event logging handler be registered with a plurality of event handlers in a pre-boot environment. The cited reference shows no such element. It will be understood by one of ordinary skill in the art that the registration of an event handler is not the same as using a predefined interrupt service routine. Further, registering the event handler is not the same as setting a pointer to a buffer. Applicants describe registering the event logging handler at least in paragraph [25] as:

> "[0025] An event logging handler is registered by the BIOS. This registration allows a protocol, or means by which event handlers or other processes in the system can call a common procedure for a common purpose. The registered handler is to be called by other processes, modules or drivers. The event logging handler is registered with the firmware that notifies all other event handlers to call this registered handler when they exhibit events. Registering an event handler installs it as a common catch-all for events that might occur. If a logging event occurs, then the event handler catches the event and logs data into the memory-based repository, or buffer, via the instructions in the event logging handler. Logging continues until pre-boot is complete."

Applicants respectfully submit that the Examiner has failed to identify "the factual basis for its rejection", as required by the Federal Circuit. The Examiner's allegations do not rise to

the level of meeting the requisite burden of proof. The rejection of Claims 1-4, 9-10, 22-24, 29-30, 42 and 45 is thus facially deficient for at least these reasons.

Further, the Examiner asserts that Treu teaches *storing the retrieved plurality of event data in a memory location accessible by an operating system, the storing being performed prior to launching of the operating system.* In fact, Treu teaches that error log is available to the operating system only through the BIOS interrupt handling service. Applicants recite a system where the memory location is available to the operating system. Treu actually teaches away from this access, at least at col. 5, lines 4-6; col. 6, lines 56-59. Treu teaches that the firmware (BIOS) acts as the interface between the OS and the error log. The OS cannot gain access to the error log without a predefined interrupt acting on a predefined location. In contrast, Applicants' invention requires that the event data is accessible to the operating system, even if stored before the OS is launched.

As for Claims 2 and 23, the Examiner uses Treu to describe that the vital product data (VPD) are on the firmware and are not accessible to the OS, but fails to show the elements of registering the event logging handler or that the event data is stored in a memory location that is accessible to the OS.

As for Claim 3, the Examiner uses Treu to describe that the proprietary memory is non-volatile memory, but fails to show the elements of registering the event logging handler or that the event data is stored in a memory location that is accessible to the OS.

As for Claims 4 and 24, the Examiner leaps to the conclusion that Treu must teach a unique identifier as "inherent to the system as a unique identifier will be needed to access the memory location" and that the unique identifier is a globally unique identifier (GUID). This assertion is completely without basis. One of ordinary skill in the art will understand that a GUID is a term of art defining a specific kind of identifier and not just merely any kind of pointer. Appendix A shows various definitions of a GUID as exist on the public Internet. Applicants assert that the term GUID was known and understood at the time of filing the application. Further, the Examiner fails to point to even a reference for a "unique identifier" that is used by Treu to access the memory location. In fact, a unique id is not taught or suggested by Treu to point to the error log. As Treu discusses, the error log area is predefined and accessed via a predefined interrupt handler. Thus, no unique identifier pointer is necessary.

As for Claims 9 and 29, the Examiner asserts that Treu teaches that *event data comprises: progress data, status data, error logging information, and general system information*, at col. 6, lines 35-55. In fact, the cited reference teaches that:

> "VPD 92 is provided for products supporting the VPD function by including a ROM containing VPD information for a particular product. Such information is read from the product ROM into VPD 92 when the system is first powered up after the product adapter has been installed. Thus, relative to FIG. 1, when token ring adapter 52 is installed, the VPD data in ROM 56 is read into VPD 92. Specific VPD information includes a unique ID for each product type or model, a part number of a field replaceable unit (FRU) for ordering such part, a replaceable unit part number identifying the part for maintenance tracking, serial number, manufacturer ID, equipment categories, resource characteristics of functions associated with product or an indication that the resources are not known or not supported, engineering change level of FRU, device driver level, and diagnostic level. VPD 92, in addition to containing such information for each product, also contains an adapter ID, a system unique ID including model/submodel and BIOS revision level. Other information could also be included as required."

It is unknown how the VPD information as described by Treu even remotely resembles event data comprising progress data, status, data, error logging information and general system information. Thus, this rejection is improper and must be withdrawn.

As for Claims 10 and 30, the Examiner asserts that Treu teaches an operating system agent. In fact, Treu only teaches that the memory-based buffer is accessible by the BIOS via an interrupt handler. At no time does Treu teach that an agent of the operating system has access to the event data in the memory buffer.

Claims 5-8, 25-28, 43-45 and 48-50 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Treu in view of Applicant Admitted Prior Art (hereinafter, "AAPA"). This rejection is respectfully traversed and Claims 5-8, 25-28, 43-45 and 48-50 are believed allowable based on the foregoing and following discussion.

The Examiner completely misunderstands Applicants' description of the invention. At no time do Applicants admit that EFI and ACPI are well known in the art (citing page 8, lines 5-8) as relating to the claimed invention. Applicants describe that system memory has reserved locations that may be used to implement the present invention. Specifically, Applicants describe: "The system memory has a reserved location for a configuration table 360, typically for an EFI Configuration Table. It will be apparent to one skilled in the art that storage in available memory

within an ACPI (Advanced Configuration Power Interface) table may be used as well." Based on the description of the claimed invention, those skilled in the art will understand that ACPI tables may be used as an alternative to the EFI tables as system memory that may be used to hold the event data. At no time have the Applicants stated or implied that the use of EFI or ACPI tables have been used in this way or that it would be obvious to modify existing systems in this way. Applicants only state, in the context of describing the invention, that it will be obvious based on the description that ACPI tables may be used. Without Applicants' description, the use of ACPI tables would not be obvious. The Examiner is not permitted to use hindsight to make an improper rejection. It is unknown how the Examiner has made this leap from description of the invention in the Detailed Description of the Specification to admitted prior art. Therefore, this rejection is improper and must be withdrawn.

As for Claims 43 and 48, the Examiner asserts that the AAPA teaches that pre-boot memory store is flash RAM. However, the Examiner fails, at least, to show a cited reference that teaches *a memory for storing event data in a pre-boot environment operatively coupled with the processor; a memory-based buffer for shadowing pre-boot environment event data, the memory-based buffer being accessible by an operating system agent, and operatively coupled to the processor; and an event logging handler running on the processor during pre-boot, the event logging handler for registering the pre-boot event data to the memory-based buffer, wherein the event logging handler is to be registered with a plurality of event handlers in the pre-boot environment.* Specifically, the cited references fail to show that the memory-based buffer is accessible to the operating system agent or that the event logging handler is registered during pre-boot.

As for Claim 49 the Examiner asserts that Treu teaches that memory for storing event data reside on the flash RAM. However, the Examiner fails to cite a reference teaching at least that the shadowed memory-based buffer is accessible to the operating system.

As for Claim 50, the Examiner asserts that Treu teaches that memory for storing event data is a firmware hub comprising a BIOS and VPD. However, the Examiner fails to cite a reference teaching at least that the shadowed memory-based buffer is accessible to the operating system.

Claims 11-12, 15-16, 19, 31-32, 35-36, 39 and 46 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Treu in view of USPN 6,785,893 to Morris et al. (hereinafter, "Morris et al."). This rejection is respectfully traversed and Claims 11-12, 15-16, 19, 31-32, 35-36, 39 and 46 are believed allowable based on the foregoing and following discussion.

As for Claims 15 and 35, as discussed above, Treu fails to teach or suggest *retrieving event data from a memory-based buffer, by an operating system agent during runtime, the memory-based buffer being generated in a pre-boot environment by an event logging handler, wherein the memory-based buffer resides in a reserved portion of system memory known to both the pre-boot environment and the runtime environment*. Treu teaches that an error log is available to the firmware only. An interrupt handler in the BIOS is used to retrieve any error data. In contrast, Applicants' recited invention requires an operating system agent to retrieve the event data at runtime from a reserved portion of system memory known to both the pre-boot environment and the OS. It will be clear to one of ordinary skill in the art that system memory is not the same as flash memory having BIOS or VPD, as taught by Treu.

Further, the Examiner asserts that Morris et al. teach displaying the event data at runtime. However, the use of the teachings by Morris et al. are incorrectly applied to Treu and combination of these two references is not only improper, but would not result in Applicants' claimed invention. Morris et al. teach that an operating system tracks events processed by interrupts and non-interrupt events. Morris et al. do not teach or imply that event data generated during pre-boot may be retrieved or displayed. Morris et al. teach a system that functions only under an OS and is unrelated to Applicants' claimed invention. Further, Morris et al. is not correctly applied to Treu who teaches that the BIOS interface is independent of the operating system. Morris et al. do not teach or suggest that event data as generated during per-boot may be displayed during runtime. Therefore, this rejection improper and should be withdrawn.

As for Claims, 11-12 and 31-32, the Examiner asserts that Morris et al. teach that *an action is to be performed by the operating system agent, in response to data accessed from the memory-based buffer*. However, at no time do Morris et al. teach or suggest that an operating system agent accesses data in a memory-based buffer where the retrieved plurality of event data in a memory location accessible by an operating system, the storing being performed prior to launching of the operating system. Morris et al. teach only event logging that occurs after OS

launch. Thus, Morris et al. cannot display event data as defined and claimed by Applicants invention. Moreover, Treu fails to teach or suggest the other claimed elements, as described above.

As for Claims 16 and 36, as discussed above, Treu does not teach the recited globally unique identifier (GUID). Nor does True teach all of the other recited elements, as discussed above. Further, Morris et al. fail to teach or suggest displaying event data that may be generated during pre-boot. Thus, the combination of Treu and Morris et al. is improper. Even when combining Treu and Morris et al., Applicants' invention would not result, as Morris et al. do not display event data generated during pre-boot and Treu does not access a reserved portion of system memory accessible to both the pre-boot environment and the OS.

As for Claims 19, 39 and 46, the Examiner misapplies Morris et al. to Treu at least as discussed above for Claims 15 and 35.

Claims 17-18 and 37-38 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Treu, Morris et al., and further in view of AAPA. This rejection is respectfully traversed and Claims 17-18 and 37-38 are believed allowable based on the foregoing and following discussion.

As discussed above, Applicants have not admitted that EFI and ACPI are well known by those of skill in the art as applied to the teaching of the claimed invention. Use of reserved system memory in the form of EFI or ACPI tables is not known to have been used to store event data as a data bridge from pre-boot to OS. Nor have Applicants inadvertently admitted or implied that this is the case. As the Examiner has failed to put forth a prima facie case of obviousness, this rejection is improper and must be withdrawn.

Claims 13-14, 20-21, 33-34, 40-41 and 47 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Treu, Morris et al., and further in view of Kimoto et al. (U.S. Patent Application Publication 2002/0184366) (hereinafter "Kimoto et al."). This rejection is respectfully traversed and Claims 13-14, 20-21, 33-34, 40-41 and 47 are believed allowable based on the foregoing and following discussion.

The Examiner asserts that Treu and Morris et al. teach or suggest each element of the claims aside from using XML for viewing data logs. As discussed above, this assertion is incorrect. Further, the application of the teaching of Kimoto et al. to Treu and Morris et al. is improper. Kimoto et al. teach a system for logging client information and transmitting to a server

side. Kimoto et al. do not suggest that information logged during pre-boot could be displayed or transmitted at runtime using XML. The actions taught by Kimito et al. occur during runtime. Thus, for the foregoing reasons, the combination of Treu, Morris et al., and Kimoto et al, is not only improper, but will not result in Applicants' claimed invention.

All claims remaining in the application are now allowable.

CONCLUSION

In view of the foregoing, Claims 1 to 50 are all in condition for allowance. If the Examiner has any questions, the Examiner is invited to contact the undersigned at (703) 633-6845. Early issuance of Notice of Allowance is respectfully requested. Please charge any shortage of fees in connection with the filing of this paper, including extension of time fees, to Deposit Account 02-2666 and please credit any excess fees to such account.

Respectfully submitted,

Dated: 10 July 2006

S/ Joni D. Stutman-Horn /
Joni D. Stutman-Horn, Reg. 42,173
Patent Attorney
Intel Corporation
(703) 633-6845

c/o Blakely, Sokoloff, Taylor &
Zafman, LLP
12400 Wilshire Blvd.
Seventh Floor
Los Angeles, CA 90025-1026

APPENDIX A: GUID Definitions

Pages from URLs:

http://en.wikipedia.org/wiki/GUID

http://www.webopedia.com/TERM/G/GUID.html

MENU
Home
Term of the Day
New Terms
Pronunciation
New Links
Quick Reference
Did You Know?
Categories
Tech Support
Webopedia Jobs
About Us
Link to Us
Advertising

Compare Prices
go
HardwareCentral

Talk To Us
Submit a URL
Suggest a Term
Report an Error

YAHOO! shopping

internet.com
Developer
International
Internet Lists
Internet News
Internet Resources
IT
Linux/Open Source
Personal Technology
Small Business
Windows Technology
xSP Resources
Search internet.com
Advertise
Corporate Info
Newsletters
Tech Jobs

GUID

Last modified: Thursday, June 20, 2002

Short for *Globally Unique Identifier*, a unique 128-bit number that is produced by the Windows OS or by some Windows applications to identify a particular component, application, file, database entry, and/or user. For instance, a Web site may generate a GUID and assign it to a user's browser to record and track the session. A GUID is also used in a Windows registry to identify COM DLLs. Knowing where to look in the registry and having the correct GUID yields a lot information about a COM object (i.e., information in the type library, its physical location, etc.). Windows also identifies user accounts by a username (computer/domain and username) and assigns it a GUID. Some database administrators even will use GUIDs as primary key values in databases.

GUIDs can be created in a number of ways, but usually they are a combination of a few unique settings based on specific point in time (e.g., an IP address, network MAC address, clock date/time, etc.).

•E-mail this definition to a colleague•

Sponsored listings

Endress+Hauser: guided radar - Supplies industrial automation and control, test and measurement, and information system solutions.

Hoover's: Manufacturing Equipment - Provides in-depth business and public/private company information, including profiles, industry overviews, financials, officers, competitors and news.

W.C. Branham: Cylinders - Makes pneumatic rodless cylinders; magnetically coupled, cable, band, low pressure hydraulic and electric. Many bore sizes and long stroke lengths.

*For internet.com pages about **GUID** **CLICK HERE**. Also check out the following links!*

LINKS = Great Page!

Sponsored listings

Bytewise: Web Guiding Systems - Offers laser-based system for tracking a geometric feature for the purpose of guiding a component during an automated process.

PSI Water Systems: Manufacturing Equipment - Designs, manufactures, sells, services, and supports the ENCON line of wastewater evaporators, drum evaporators, and distillation systems.

Flexible Assembly Systems: Process Equipment - A leading supplier of ergonomic assembly tools, handling devices, torque products & systems for production needs.

ITT Flygt: Manufacturing Equipment - Supplies submersible pumps, mixers, and accessories for uses including water management and treatment, industrial effluent handling, and dewatering.

Manufacturing Equipment: CSC Force Measurement - Provides force gauge testing machines for tensile/compression testing. Includes grips, fixtures and software for data collection and machine control.

Related Categories

Databases

Windows

Related Terms

COM

DLL

(Webopedia)
Give Us Your Feedback

Maximize the benefits of anti-piracy solutions and minimize revenue loss.	Learn why a third party software licensing solution can can be the best option for your company.	How can you make it more expensive for users to pirate your software than to purchase it? Find out!

Globally Unique Identifier

From Wikipedia, the free encyclopedia
(Redirected from GUID)

A **Globally Unique Identifier** or **GUID** is a pseudo-random number used in software applications. While each generated GUID is not guaranteed to be unique, the total number of unique keys (2^{128} or 3.4028×10^{38}) is so large that the possibility of the same number being generated twice is very small.

GUIDs are used in many pieces of software, including Oracle Database and Novell eDirectory, but the most high-profile GUID implementation may be Microsoft's. There is a standard called Universally Unique Identifier (UUID), specified by the Open Software Foundation (OSF).

Contents

Basic structure

The GUID (pronounced gwid (as in Squid) especially by Microsoft; alternate pronunciation goo-id) is a 16-byte (128-bit) number, written in hexadecimal form, such as:

3F 25 04 E0 4F 89 11 D3 9A 0C 03 05 E8 2C 33 01

While a GUID strictly has no formal substructure, they may be written in text in varying ways depending on the implementation. In one such method GUIDs are written using the hexadecimal representation of a four-byte word, 2 two-byte words, and a eight-byte word separate by hyphens, such as:

{3F2504E0-4F89-11D3-9A0C0305E82C3301}

However, the most commonly used structure of the data type is:

```
GUID STRUCT
Data1 dd
Data2 dw
Data3 dw
Data4 db 8
GUID ENDS
```

The definition of guid from guidef.h is as shown below:

```
typedef struct _GUID {
    unsigned long  Data1;
    unsigned short Data2;
    unsigned short Data3;
    unsigned char  Data4[ 8 ];
} GUID;
```

Using the above structure definitions, a hexadecimal representation could also be:

{3F2504E0-4F89-11D3-9A-0C-03-05-E8-2C-33-01}

In the Microsoft component object model, GUIDs are used to uniquely distinguish different software component interfaces. This means that two (possibly incompatible) versions of a component can have exactly the same name but still be distinguishable by their GUIDs.

GUIDs are also inserted into documents from Microsoft Office programs, as these are regarded as objects as well. Even audio or video streams in the Advanced Streaming Format (ASF) are identified by their GUIDs.

In Advanced Streaming Format (ASF) files at least, and probably in general, the GUID data is stored in little endian format as a 32-bit unsigned integer, followed by 2 16-bit unsigned integers, followed by 8 unsigned bytes. Software on hardware with a big endian CPU must reverse the bytes in the first 32-bit, and both 16-bit quantities, the remaining 8 bytes are fine as is. (The display format is somewhat misleading.)

Algorithm

The OSF-specified algorithm used by Microsoft for generating new GUIDs has been widely criticized. In these (V1) GUIDs, the user's network card MAC address was used as a base for the last group of GUID digits, which meant, for example, that a document could be tracked back to the computer that created it. This privacy hole was used when locating the creator of the Melissa worm.

V1 GUIDs which contain a MAC address can be identified by the digit "1" in the first position of the third group of digits, for example {2f1e4fc0-81fd-*1*1da-9156-00036a0f876a}. GUIDs using the later algorithm, which has a random suffix have a "4" in the same position, for example {38a52be4-9352-*4*53e-af97-5c3b448652f0}.

Subtypes

There are several flavors of GUIDs used in COM:

- IID - interface identifier;
- CLSID - class identifier;
- LIBID - type library identifier;
- CATID - category identifier; (its presence on a class identifies it as belonging to certain class categories)

DCOM introduces many additional GUID subtypes:

- AppID - application identifier;
- MID - machine identifier;
- IPID - interface pointer identifier; (applicable to an interface engaged in RPC)
- CID - causality identifier; (applicable to a RPC session)
- OID - object identifier; (applicable to an object instance)
- OXID - object exporter identifier; (applicable to an instance of the system object that performs RPC)
- SETID - ping set identifier; (applicable to a group of objects)

These GUID subspaces may overlap, as the context of GUID usage defines its subtype. For example, there might be a class using same GUID for its CLSID as another class is using for its IID — all without a problem. On the other hand, two classes using same CLSID couldn't co-exist.

XML syndication formats

There is also a `guid` tag in some versions of the RSS specification, and mandatory `id` tag in Atom, which should contain a unique identifier for each individual article or weblog post. In RSS the contents of the guid can be any text, and in practice is typically a copy of the post URL. Atom's IDs need to be valid URIs (usually URLs pointing to the entry, or URNs containing any other unique identifier).

External links

- DmaId for InstanceId Values (DCE Universally Unique IDentifiers, UUIDs) (http://www.infonuovo.com/dma/csdocs/sketch/instidid.htm)
- Syntax and semantics of the DCE variant of Universal Unique Identifiers (UUIDs) (http://www.opengroup.org/onlinepubs/9629399/apdxa.htm)
- Draft UUID specification (includes sample code) (http://www.webdav.org/specs/draft-leach-uuids-guids-01.txt)
- A Universally Unique IDentifier (UUID) URN Namespace (http://www.ietf.org/rfc/rfc4122.txt)
- UUID (GUID) Generator on the Web (http://kruithof.xs4all.nl/uuid/uuidgen)

Retrieved from "http://en.wikipedia.org/wiki/Globally_Unique_Identifier"

Categories: Microsoft Windows | Identifiers
